REMARKS

Applicant amended independent claim 13 to add the clarifying feature, appearing in independent claims 26 and 31, that the processing engines are each "multi-threaded" processing engines, and to further clarify that each of the processing engines is configured to execute separate groups of threads. Applicant also amended independent claim 13 to clarify that the detected signal is generated in response to the request from the resource shared by the multiple multi-threaded processing engines. Applicant amended independent claims 26 and 31 to recite features similar to those recited in amended claim 13. After these amendments, claims 13-18, and 20-35 are pending. Claims 13, 26 and 31 are independent.

The examiner rejected claims 13-18 20, 21 and 24 under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 6,295,600 to Parady in view of U.S. Patent No. 6,507,862 to Joy. The examiner further rejected claims 22 and 23 under 35 U.S.C. §103(a) as being unpatentable over Parady in view of Joy. Additionally, the examiner rejected claim 25 under 35 U.S.C. §103(a) as being unpatentable over Parady in view of Joy, and further in view of U.S. Patent No. 6,085,215 to Ramakrishnan. The Examiner also rejected claims 26-35 under 35 U.S.C. §103(a) as being unpatentable over Ramakrishnan, in view of Parady and Joy.

Specifically, with respect to applicant's independent claim 13, the examiner stated:

> 2. Parady taught the invention substantially as claimed including a data processing ("DP") system comprising: multiple processing engines (32,34,36,38,40,42,44,46, see fig. 1) executing a least one instruction of a first thread having a first program counter[addresses in the program address registers](see. col. 3. lines 58-65), the at least one instruction including at least one instruction to issue request to a resource shared by the multiple processing engine (e.g., see figs. 1,2 and col. 3, line 8-col. 4, line 62);
> 3. Swapping execution to a second thread having a second program counter after processing engine execution of the at least one instruction to issue the request to the shared resource (e.g., see col. 4, line 3-col. 5, line 5); and
> 4. Swapping execution to the first thread after detection of a signal generated in response to the request to the shared resource (e.g., see col. 4, lines 52-62).
> 5. Parady did not expressly detail individual one of execution engines including corresponding arbiters to select threads for execution. Joy however taught individual ones of multithreaded execution engines (904,902) including arbiters (e.g., see col. 10, lines 26-40 and col. 13, lines 8-65)[each multithreaded pipeline has its own thread selectable flip-flop substitution logic used to create vertically multithreaded functionality].

> 6. One of ordinary skill in the DP art would have been motivated to combine the teachings of Parady and Joy. Parady taught processor the processed multiple threads and switched on a long latency event such as a cache miss to allow processing to continue on long latency events(e.g., see col. 2, lines 25-42). Joy taught the improvement of efficiency and parallel by introducing multithreading in two steps , vertical multithreading and horizontal multithreading (e.g., see col. 3, lines 3-13). Therefore one of ordinary skill would have been motivated to incorporate the Joy vertical and horizontal multithreading into the processor that already processed multiple threads to increase efficiency so that time threads would be processed in a timely manner when a long latency even occurred such as a cache miss.
>
> As to the further limitation of claim 13, Parady taught executing one or more additional instructions of the first thread, after executing the at least one instruction (load or store) to issue the request to the shared resource (e.g., see col. 1, lines 50-62 and col. 4, line 63-col. 5, line 5)[for non blocking loads, which allow further instructions in the same thread after the load to be executed, with the system remaining and executing in the first thread. Parady taught the non-blocking load typically later becomes a blocking load where at a later point the execution of further instructions is stopped while waiting for the load or store to complete and at that later point or later thread swapping is allowed]. (Final Action, page 2-4)

Applicant respectfully disagrees with the examiner's contentions, particularly the contentions expressed in paragraphs 2-4 of the Final Action.

Applicant's independent claim 13 recites "[a] method, comprising: at a multi-threaded processing engine within a processor having multiple multi-threaded processing engines that are each configured to execute separate groups of threads, individual ones of the engines including corresponding arbiters to select from the corresponding set of threads for execution, executing at least one instruction of a first thread having a first program counter, the at least one instruction including at least one instruction to issue a request to a resource shared by the multiple multi-threaded processing engines; ...; and swapping execution to the first thread after detection of a signal generated in response to the request to the resource shared by the multiple multi-threaded processing engines." Thus, applicant's multiple processing engines each process separate groups of multiple threads. Additionally, after a particular thread executing at one of the processor issues a request to a resource shared by the multiple processing engines and is swapped out, the thread is swapped back in after detection of a signal from the resource shared by the multiple multi-threaded processing engines.

In contrast, Parady describes apparatus for switching between threads of a program in response to a long-latency event (Abstract). Parady describes its apparatus as follows:

> **FIG. 1 is a block diagram of an UltraSparc™ microprocessor 10, modified to incorporate the present invention. An instruction cache 12 provides instructions to a decode unit 14. The instruction cache can receive its instructions from a prefetch unit 16, which either receives instructions from branch unit 18 or provides a virtual address to an instruction TLB (translation look-aside buffer) 20, which then causes the instructions to be fetched from an off-chip cache through a cache control/system interface 22. The instructions from the off-chip cache are provided to a pre-decode unit 24 to provide certain information, such as whether it is a branch instruction, to instruction cache 12.**
>
> **Instructions from decode unit 14 are provided to an instruction buffer 26, where they are accessed by dispatch unit 28. Dispatch unit 28 will provide four decoded instructions at a time along a bus 30, each instruction being provided to one of eight functional units 32-46. The dispatch unit will dispatch four such instructions each cycle, subject to checking for data dependencies and availability of the proper functional unit.**
>
> **The first three functional units, the load/store unit 32 and the two integer ALU units 34 and 36, share a set of integer registers 48. Floating-point registers 50 are shared by floating point units 38, 40 and 42 and graphical units 44 and 46. Each of the integer and floating point functional unit groups have a corresponding completion unit, 52 and 54, respectively. The microprocessor also includes an on-chip data cache 56 and a data TLB 58.**
>
> **FIG. 2 is a block diagram of a chipset including processor 10 of FIG. 1. Also shown are L2 cache tags memory 80, and L2 cache data memory 82. In addition, a data buffer 84 for connecting to the system data bus 86 is shown. In the example shown, a 16-bit address bus 88 connects between processor 10 and tag memory 80, with the tag data being provided on a 28-bit tag data bus 89. An 18-bit address bus 90 connects to the data cache 82, with a 144 bit data bus 92 to read or write cache data.**
>
> **FIG. 3 illustrates portions of the processor of FIG. 1 modified to support the present invention. As shown, a decode unit 14 is the same as in FIG. 1. However, four separate instruction buffers 102, 104, 106 and 108 are provided to support four different threads, threads 0-3. The instructions from a particular thread are provided to dispatch unit 28, which then provides them to instruction units 41, which include the multiple pipelines 32-46 shown in FIG. 1. (emphasis added, col. 3, lines 7-52)**

Thus, Parady's apparatus includes a single processing engine to process a single group of threads. As such, the available resources are used only by that one processing engine, and are not shared by multiple multi-threaded processing engines that are each configured to execute separate groups of threads. Accordingly, Parady does not disclose or suggest at least the feature of "at a multi-threaded processing engine within a processor having multiple multi-threaded

processing engines that are each configured to execute separate groups of threads, individual ones of the engines including corresponding arbiters to select from the corresponding set of threads for execution," as required by applicant's independent claim 13.

Moreover, since Parady's apparatus does not have resources that are shared by multiple multi-threaded processing engines, Parady's apparatus neither has threads that issue requests to such shared resources nor does the apparatus swap execution to a swapped-thread after detection of a signal generated in response to a request to such shared resources. Indeed, no such shared resources exist. Accordingly, Parady fails to disclose or suggest at least the feature "executing at least one instruction of a first thread having a first program counter, the at least one instruction including at least one instruction to issue a request to a resource shared by the multiple multi-threaded processing engines; ...; and swapping execution to the first thread after detection of a signal generated in response to the request to the resource shared by the multiple multi-threaded processing engines," as required by applicant's independent claim 13.

As for the examiner's contentions that Parady's elements 32, 34, 36, 38, 40, 42, 44 and 46 correspond to multiple processing engines, as described in Parady, the units 32-46 are functional units of a single processor 10, and include "the load/store unit 32 and the two integer ALU units 34 and 36, share a set of integer registers 48. Floating-point registers 50 are shared by floating point units 38, 40 and 42 and graphical units 44 and 46" (FIG. 1, col. 3, lines 27-30). Thus, the functional units 32, 34, 36, 38, 40, 42, 44 and 46 are modules within a processor configured to perform specific functions in the course of processing instructions received by the processor (e.g., the Integer ALU 36 would presumably perform ALU operations on integer operands specified in the received instructions). These functional units perform operations only in relation to threads 0-3 executing on the single processor 10. Thus, the functional units 32, 34, 36, 38, 40, 42, 44 and 46 do not correspond to multiple processing engines that are each configured to execute separate groups of threads.

Joy describes multi-threading processors (col. 1, lines 37-39.) In relation to a single-processor vertically threaded processor 300 (shown in FIG. 3), Joy describes that thread switching is implemented using high speed multiple flip-flops (see FIG. 4 and accompanying description at col. 10, line 41 to col. 13, line 7), and a thread switch logic 610 that at least in part controls the flip-flops. Particularly, Joy explains that:

> **The fast, nanoseconds range context switch operates in conjunction with thread switching logic such as the pulse-based high-speed flip-flop 400 to improve speed of thread switching. The pulse-based high-speed flip-flop 400 enables virtually instantaneous switching between threads, saving of the machine state of a stalled thread, and machine state restoration of an activated thread. The fast, nanoseconds range, context switching rapidly controls which thread is activated by the pulse-based high-speed flip-flop 400. The thread switch logic 610 receives a plurality of input signals that evoke a context switch and thread switch. In an illustrative processor, input terminals to the thread switch logic 610 include an L1_load_miss terminal, an L1_instruction_miss terminal, an instruction_buffer_empty terminal, a thread_priority terminal, an MT_mode terminal, an external_interrupt terminal, and an internal_interrupt terminal. The thread switch logic 610 generates a thread identifier (TID) signal based on signals to the input terminals. The thread switch logic 610 generates the TID signal with a thread switch delay or overhead of one processor cycle. (Joy, col. 15, line 52, to col. 16, line 5)**

And,

> **In other implementations, the thread switch logic 610 implements one or more of several thread-switching methods. ...**
> ...
> **A third thread-switching operation is an "intelligent global scheduler" thread-switching in which a thread switch decision is selectively programmed, based on one or more signals. In one example an intelligent global scheduler uses signals such as: (1) an L1 data cache miss stall signal, (2) an L1 load miss signal, (3) an instruction buffer empty signal, (4) an instruction queue empty signal, (5) an L2 cache miss signal, (6) a thread priority signal, (7) a thread timer signal, (8) an interrupt signal, or other sources of triggering. In some embodiments, the thread select signal is broadcast as fast as possible, similar to a clock tree distribution. In some systems, a processor derives a thread select signal that is applied to the flip-flops by overloading a scan enable (SE) signal of a scannable flip-flop. (Joy, FIG. 6, and col. 16, line 66, to col. 17, line 32)**

The signals used to control thread switching in Joy's processor are all signals that are local to the multi-threaded processor in which the switch logic 610 resides. For example, the various cache signals are generated by the instruction cache L1 of the processor (see, col. 9, lines 40-44.)

Although Joy describes a processor that includes multiple vertically-threaded processors (see for example, FIG. 9, and accompanying description at col. 18, lines 39 to col. 20, line 63), at no point does Joy describe that thread swapping operations are affected by signals generated in response to requests to resources shared by all processors. Rather, thread swapping at individual vertically-threaded processors depends only on the signals generated locally at the respective

vertically-threaded processors. Accordingly, Joy fails to disclose or suggest at least the feature of "executing at least one instruction of a first thread having a first program counter, the at least one instruction including at least one instruction to issue a request to a resource shared by the multiple multi-threaded processing engines; ...; and swapping execution to the first thread after detection of a signal generated in response to the request to the resource shared by the multiple multi-threaded processing engines," as required by applicant's independent claim 13.

Because neither Parady, nor Joy, discloses or suggests, alone or in combination, at least the feature of "executing at least one instruction of a first thread having a first program counter, the at least one instruction including at least one instruction to issue a request to a resource shared by the multiple multi-threaded processing engines; ...; and swapping execution to the first thread after detection of a signal generated in response to the request to the resource shared by the multiple multi-threaded processing engines," applicant's independent claim 13 is patentable over the cited art.

Claims 14-18 and 20-25 depend from independent claim 13, and are therefore patentable for at least the same reasons as independent claim 13.

As noted, the examiner rejected claims 26-35 under 35 U.S.C. §103(a) as being unpatentable over Ramakrishnan, in view of Parady and Joy.

Amended independent claims 26 and 31 recite "execute at least one instruction of a first thread having a first program counter, the at least one instruction including at least one instruction to issue a request to a resource shared by the multiple multi-threaded processing engines, ..., and swap execution to the first thread after detection of a signal generated in response to the request to the resource shared by the multiple multi-threaded processing engines," or similar language. For reasons similar to those provided with respect to independent claim 13, at least these features are not disclosed or suggested by Parady and/or Joy.

Ramakrishnan describes method and apparatus for avoiding receive livelock and transmit starvation, and for minimizing packet loss and latency in a communication network station (Abstract). To that end, Ramakrishnan explains that "[b]riefly, and in general terms, the method of the invention comprises the steps of dividing processing tasks into processing threads, each of which is structured to execute for a limited time before being subject to preemption by another

processing thread" (col. 4, lines 16-20). In particular, Ramakrishnan's apparatus uses a scheduler to control execution of the threads. Ramakrishnan explains:

> **The core thread scheduler 46 is a special task to which control is transferred at the end of each processing thread. The scheduler is simply a round-robin selection device with knowledge of all of the processing threads and access to the real time thread flags 50. When one processing thread is completed, the scheduler 46 chooses the next one in turn having a flag set to indicate that processing is needed. FIG. 5 shows the basic steps performed by the core thread scheduler 46. After return from a completed processing thread, indicated at 56, the scheduler 46 selects the next thread that needs to run (block 58), and yields control to the next thread (block 60). The scheduler 46 includes at least one time slot in its round robin system for the general purpose domain 42. (col. 9, lines 9-22)**

Thus, Ramakrishnan's apparatus includes a single multi-thread processor with a single thread scheduler 46 that implements a round-robin thread selection scheme. Ramakrishnan neither describes nor suggests multiple multi-threaded processing engines that each execute separate groups of threads, and nowhere does Ramakrishnan describe that requests are made to resources shared by such multiple multi-threaded processing engines. Indeed, there are no multiple processing engines described in Ramakrishnan. Ramakrishnan, therefore, also does not describe that swapping of execution to a swapped-out thread occurs after detection of a signal generated in response to a request to a resource shared by multiple multi-threaded processing engines. Accordingly, Ramakrishnan fails to disclose or suggest at least the feature of "execute at least one instruction of a first thread having a first program counter, the at least one instruction including at least one instruction to issue a request to a resource shared by the multiple multi-threaded processing engines, …, and swap execution to the first thread after detection of a signal generated in response to the request to the resource shared by the multiple multi-threaded processing engines."

Because none of Parady, Joy, and Ramakrishnan, discloses or suggests, alone or in combination, at least the feature of "execute at least one instruction of a first thread having a first program counter, the at least one instruction including at least one instruction to issue a request to a resource shared by the multiple multi-threaded processing engines, …, and swap execution to the first thread after detection of a signal generated in response to the request to the resource shared by the multiple multi-threaded processing engines," applicant's independent claims 26 and 31 are patentable over the cited art.

Claims 27-30 depend from independent claim 26 and are therefore patentable for at least the same reasons as independent claim 26. Claims 32-35 depend from independent claim 31 and are therefore patentable for at least the same reasons as independent claim 31.

It is believed that all the rejections and/or objections raised by the examiner have been addressed.

In view of the foregoing, applicant respectfully submits that the application is in condition for allowance and such action is respectfully requested at the examiner's earliest convenience.

Canceled claims, if any, have been canceled without prejudice or disclaimer. Any circumstance in which the applicant has (a) addressed certain comments of the examiner does not mean that the applicant concedes other comments of the examiner, (b) made arguments for the patentability of some claims does not mean that there are not other good reasons for patentability of those claims and other claims, or (c) amended or canceled a claim does not mean that the applicant concedes any of the examiner's positions with respect to that claim or other claims.

Please apply any required fees to deposit account 06-1050, referencing the attorney docket number shown above.

Respectfully submitted,

Date: Dec. 14, 2006

Ido Rabinovitch
Attorney for Intel Corporation
Reg. No. L0080

PTO Customer No. 20985
Fish & Richardson P.C.
Telephone: (617) 542-5070
Facsimile: (617) 542-8906